

# RadSrc Library and Application Manual

The RadSrc development team  
Lawrence Livermore National Laboratory

February 22, 2007

## Abstract

The RadSrc (“Rad-Source”) suite provides computational support for applications addressing radioactive decay and emission of radiation from decay. The RadSrc library, *libradsrc*, computes the concentrations of decay products given an initial concentration and age, and photon radiation due to continuing decay of those products. Written in C++, *libradsrc* provides an object-oriented interface to computational results, as well as its underlying database of isotope information. *Libradsrc* also provides a simplified interface in FORTRAN and C++ intended for use in Monte Carlo applications, and can accommodate varying levels of integration with other code bases. A stand-alone application, *gsource*, serves as an interactive user interface to the library. The RadSrc suite is open source and licensed under the BSD license and can be downloaded from <http://nuclear.llnl.gov/CNP/simulation>.

## 1 Introduction

Many applications exist that require the calculation of an ideal theoretical radiation spectrum resulting from the natural decay of radioactive elements. Often this idealized source spectrum is modified through Monte Carlo simulation of radiation transport to account for absorption and scattering of radiation in matter. Three such Monte Carlo transport codes, MCNP/X[1], GEANT4[2], and COG[3] allow the user to specify custom radiation sources in the transport simulation. This nevertheless required users to manually specify the radiation source distributions and/or supply samples from the distributions.

A previous solution, GAMGEN[4], automated the calculation of decay product concentrations in an aged material containing radioisotopes and the distribution of photons emitted by nuclear decay. Despite the added convenience of having the photon distributions computed, manual intervention was required to communicate this information into the Monte Carlo codes.

The RadSrc Suite has been developed to incorporate this calculation directly into the Monte Carlo codes themselves. Written in modern C++, the RadSrc library *libradsrc* takes advantage of modern techniques to streamline the calculation and modularize the functions of database handling, decay product and photon calculations, and provide flexible interfaces. *Libradsrc* simultaneously provides a rich interface to the library capabilities for C++ applications while it also provides a slim interface to FORTRAN Monte Carlo applications. The object-oriented design facilitates the adaptation of new sources of isotope data beyond that provided in the RadSrc distribution. A stand-alone application, *gsource*, is included to reproduce the functionality of GAMGEN using the RadSrc library.

*Gsource* accepts an initial isotope mixture and desired age and computes the concentrations of the decay products, and photon emission spectrum from radioactive isotopes in the aged mixture. The user has the

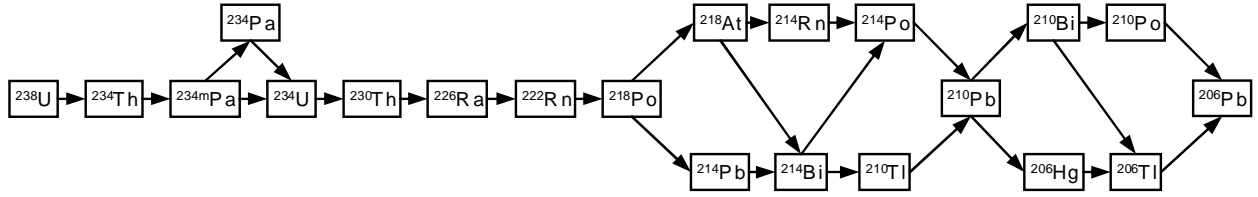


Figure 1: Decay paths for  $^{238}\text{U}$  and its daughter isotopes, including at most a single alpha and a single beta decay. There are 30 possible paths from  $^{238}\text{U}$  to  $^{206}\text{Pb}$ .

opportunity to select certain lines of interest and bin the remainder; this can simplify comparison of the computed spectrum to experimental data. Several binning options are provided, including pre-computed binning schemes TART and MORSE, uniform binning and binning proportional to a square-root of energy function. The user is similarly free to choose any of these binning schemes for the computed bremsstrahlung radiation (if present) and the custom tally bins for the MCNP and MCNP/X transport codes.

The RadSrc Suite and its data has been released under the BSD open source license. Users are free to incorporate RadSrc functionality into their own applications, or adjust and enhance the isotope database with new isotopes, new decay modes, or improved values.

## 2 Computation of Decay Products

Radioisotopes can decay into other isotopes with lower average binding energy per nucleon[5]. Isotopes will typically decay by one of several processes: *alpha decay*, which emits an  $\alpha$  particle (helium nucleus) and reduces both the atomic number and atomic mass by two; *beta-minus* decay, with emission of a  $\beta^-$  particle (electron), which increases the atomic number; or beta decay with (*beta-plus*) or without (*electron capture*) emission of a  $\beta^+$  particle (positron), either of which decreases the atomic number.

Radioactive decay is frequently accompanied by emission of photons. The parent isotope may decay to an excited state, which enter the ground state by emitting one or more gamma rays. The atomic electrons may also emit x rays as they re-adjust to the new nuclear potential. If a charged particle is emitted in the decay, it may also emit photon radiation as it travels through condensed matter, called *bremsstrahlung radiation*[5].

A radioisotope may decay by both alpha and beta modes. Also, a particular decay mode may decay to both the ground state and one or more excited states of the daughter nucleus, with different gamma and bremsstrahlung emission. These are termed *decay branches*. The probability of any particular decay branch occurring is the *branch fraction*. There are further variations of these decays. For example, a decay to an excited state may result in emission of a neutron. The rates of decay for all the decay branches sum to the overall *decay rate*. The *half-life*,  $\ln 2$  divided by the overall decay rate, is the time when half of the atoms of a radioisotope have decayed.

A radioisotope may decay into one or more isotopes that are themselves radioactive. As these daughter isotopes decay into other radioisotopes, a chain of decays is followed. This chain will branch at isotopes with multiple significant decay, and often rejoin when two paths contain the same alpha and beta decays but in a different order. The decay chain of  $^{238}\text{U}$  is shown in Figure 1 as a typical example.

As time progresses, the numbers of atoms of the daughter isotopes rise and fall. Since the half-lives of isotopes in the decay chain range over many orders of magnitude, certain isotopes will come to dominate

the decay of other isotopes, and the concentrations of these isotopes will be fixed to a nearly constant ratio. This is termed *secular equilibrium*. One may infer the presence and concentration of an isotope that does not emit photons by detecting and measuring the intensity of photons emitted by another isotope that is in secular equilibrium with the hidden isotope. To do this, we must calculate the isotope ratio at the time of measurement.

Bateman derived the closed-form solution to the differential equations governing radioactive decay through the use of Laplace transforms in 1910[6][7]. The form of the solution lends itself to calculation by recursion. Without loss of generality, we can reduce the problem of multiple initial parent isotopes to independent problems with a single parent isotope and sum the individual solutions. Each parent isotope is the root of a directed acyclic graph with decay products as nodes. Further, we can consider each path in the directed graph independently of the other paths and sum the solutions for each linear decay chain. These decompositions are made possible by the linearity of the original system of differential equations.

Consider the following system of equations describing a particular decay path:

$$N_0'(t) = -\lambda_0 N_0(t) \quad (1)$$

$$N_i'(t) = f_{i-1} \lambda_{i-1} N_{i-1}(t) - \lambda_i N_i(t) \quad (2)$$

$$N_i(0) = \begin{cases} 1 & i = 0 \\ 0 & i > 0 \end{cases} \quad (3)$$

In these expressions,  $N_i(t)$  is the time dependent concentration of isotope  $i$ ,  $\lambda_i$  is the decay constant for that isotope, and  $f_i$  is the branch probability for the particular decay of isotope  $i$  given in this chain. It is readily verified by substitution that this system has solutions of the form:<sup>1</sup>

$$N_i(t) = \sum_{j=0}^i A_{i,j} e^{-\lambda_j t} \quad (4)$$

with

$$\begin{aligned} A_{0,0} &= 1 \\ A_{i,j} &= \frac{f_{i-1} \lambda_{i-1}}{\lambda_i - \lambda_j} A_{i-1,j}; \quad 0 \leq j < i \\ A_{i,i} &= - \sum_{j=0}^{i-1} A_{i,j} \end{aligned} \quad (5)$$

This recursive formulation is particularly amenable to calculation with dynamic programming techniques. The decay graph is traversed using the lists of decay modes provided by the isotope database. Each branch is followed in turn by recursively calling the calculation routine with the branch fraction and daughter isotopes applicable to that branch, and the constants  $A_{i,j}$  for the branching isotope. Upon reaching a stable isotope, the recursive functions return to each branch point, then following the next branch until no branches remain. This avoids re-computing the constants for all the isotopes preceding a branching decay.

The pairs of constants  $A_{i,j}$  and decay constants  $\lambda_j$  are stored in a mapping as they are computed. Terms associated with the same  $\lambda_j$  are summed as all possible decay paths to a particular daughter isotope are encountered. At the conclusion of the calculation, the mapping holds all contributing exponential terms to the time dependent concentration of each isotope.

---

<sup>1</sup>This form implicitly assumes that all decay constants are distinct. Although this is true in general for nuclear decay sequences, *libradsrc* will exit if presented with a contrived example containing identical decay constants.

### 3 The RadSrc Library

Four components comprise the RadSrc library, *libradsrc*:

- isotope database management,
- decay product calculations,
- photon intensity calculations,
- and interfacing routines.

*Libradsrc* is written in C++ and employs an object-oriented design. Users who are solely interested in obtaining photon distributions for Monte Carlo simulations will be satisfied by the simplified FORTRAN and C++ interfaces presented. Other users of the library may directly interface with the library objects to access the isotope database or the full capabilities of decay product and photon calculation routines. Details of the library API are presented in the appendix.

The required information about isotopes is provided by the **CIsotopeDatabase** class. This class contains physical information about isotopes such as mass and half-life; a listing of all decay modes, their branch fractions, and daughter isotope; and a listing of all gamma rays and x rays produced by each decay. The class provides iterators to list all decays and photons in an isotope record. Isotope databases are in turn managed by the **CDatabaseManager**. The **CDatabaseManager** makes it possible to obtain isotope information from different data suppliers as well as accommodate different storage mechanisms.

Decay product calculations are handled by the **CDecayComputer** class. The object is configured with an initial atomic fractions of isotopes and the desired age. Also, an isotope database is selected. After performing the decay product calculation, the object provides a listing of a decay products, their concentrations, the decays producing the products, and the ultimate parent isotope(s) present in the initial mixture that yielded each decay product. Users may create multiple **CDecayComputer** objects to address multiple sources in the same calculation.

Photon intensity calculations are handled by the **CPhotonComputer** class. This object is associated with a **CDecayComputer** object when it is created. This object takes the aged isotope concentrations and computes the discrete x ray and gamma ray emissions listed in the isotope database for each decay mode of each isotope present. It also computes a binned bremsstrahlung distribution when applicable if the necessary data is present in the isotope database. A **CPhotonComputer** object provides a list of all the photon lines, sorted by either energy or intensity, as well as the decays that produced the photons and the ultimate parent isotope(s) in the initial mixture. This object also permits sampling from the photon distribution using a user-supplied random number generator.

Users will obtain the greatest utility of the library by directly employing the aforementioned objects. However, we anticipate that many users will simply want to obtain the aged mixture concentrations and sample photons from decay sources in the mixtures. To that end, we provide a simplified interface to both FORTRAN/C and C++. The simplified interface encapsulates the **CPhotonComputer** and **CDecayComputer** classes, handles initialization, and interfaces with the user's code subject to most of the limitations of FORTRAN 77. The ability to address independent mixture calculations is retained in the FORTRAN environment by providing an opaque handle to the user upon initialization. This handle is passed back to the library in every function call.

Initialization of the library requires special attention. Users attempting to integrate *libradsrc* into an application with controlled access to the source code may be limited in ways they can communicate information to the library. Therefore we have provided four mechanisms for initializing the library:

- Hard-coded initialization in the user's own code using `RSADDISOTOPE( )` and `RSMIX( )`.
- Passing character data that has been incorporated into the application's I/O and initialization routines on to the library one line at a time using `RSADDCONFIG( )` and `RSSOURCECONFIG( )`.
- Hard-coding the filename of a configuration file into the user's own code using `RSLOADCONFIG( )`.
- Relying on environment variables to locate both the isotope data and the problem configuration.

With these four mechanisms available, *libradsrc* can accommodate any level of integration with the user's application.

## 4 RadSrc application

The calculation of radioactive decay products, although straightforward, is well suited for automation. The RadSrc library combines radioisotope data, x-ray and gamma-ray line catalogues, and measured bremsstrahlung spectra to automatically compute the photon emission from an aged radiological sample. *Gsource* is the interactive user-interface for computing photon intensity distributions. It is designed to duplicate the output and essential functionality of the GAMGEN application, using the RadSrc library to perform the calculations.

Application features:

- Calculation of decay product concentrations given an initial isotope mixture and the age
- List of gamma and x-ray photons produced by radioactive decay, as well as the particular decay(s) generating each photon line and the initial isotope responsible for the presence of the line
- Sorted output of photons by energy and intensity per gram
- Bremsstrahlung background for the U-238 decay family
- Computes bin structure for separate tally of scattered and unscattered photons during transport simulation
- Flexible selection of binning methods for photon lines, bremsstrahlung photons, and scattered photon tally
- Outputs photon lines, bremsstrahlung photons, and tally bin structures in MCNP-compatible format
- Written in portable C++ language

### 4.1 Usage

*Gsource* is a command line application. It can be invoked interactively with the command *gsource*. *Gsource* will also accept problem information from standard input or from a file, enabling batch execution.

*Gsource* may be invoked as follows:

```
gsource -h
gsource [-q] [config file]
```

If a configuration file is specified on the command line, then that file is read to obtain the problem data. Otherwise, information is taken from stdin. The -q flag suppresses all output to stdout, including run summaries and interactive prompting. The -h flag provides command-line syntax help.

*Gsource* will first attempt to load the database. If the environment variable for the default database is present, it will look in the location specified. As shipped, *gsource* is configured to read from the GAMGEN legacy database, and checks for the RADSRC\_LEGACYDATA environment variable. If this variable is not found, *gsource* will check for the non-specific RADSRC\_DATA variable. If RADSRC\_DATA is not set, or is incorrect, *gsource* will look for a directory called "data" in the current directory. If no database can be found, *gsource* will exit with a message suggesting that the user set the RADSRC\_DATA environment variable.

*Gsource* now requires a specification of the initial isotope mixture and the desired age of the computed mixture. Isotopes are specified by their common abbreviation (case insensitive) and atomic weight e.g. U-238 or u238, but not U 238. Meta-stable isotopes have an m suffix, e.g. Pa-234m. The isotope identifier is followed by the isotope concentration as a atomic percentage. For your convenience, all of the isotopes and their concentrations may be specified as a single line, separated by whitespace (spaces, tabs, and newlines). *Gsource* will ensure that each isotope you specify exists in the RadSrc database. If you mistype an entry, you must reenter that isotope (and all following entries if entered on the same line).

When you have entered all of the initial isotopes and their concentrations, enter the keyword *age* followed by the desired age in years. *Gsource* will check the total concentration to ensure the sum is approximately 100%. The total concentration need not be exactly 100% to allow the user to specify concentrations to different precisions. However if the sum is less than 100% or greater than 100.1%, *gsource* will assume a data entry error and exit with a message.

Example:

```
>cd radsrc
>bin/gsource
```

```
Enter Sources on one or multiple lines, ending with "age xx.xx" (in years)
The total quantity must be between 100% and 100.1%
```

```
Example: U238 95.0 U235 5.0 Age 10
```

```
Example: Pa-234m 5e-14
```

```
        Pa-234 2e-14
```

```
        U-238 1.0
```

```
        Age 10
```

```
Enter <isotope> <percent> or AGE <years> --> Pb-210 100 age 10
```

```
Input Composition: (fractional units)
```

```
Pb-210 1
```

```
Total: 1
```

```
Aged Composition: (fractional units)
```

```
Hg-206 1.12057e-12
```

```
Tl-206 9.24168e-13
```

```
Pb-206 0.253889
```

```
Pb-210 0.732984
```

```
Bi-210  0.000451401
Po-210  0.0126763
Total:  1
```

After entering the problem specification, *gsource* will compute the concentration of isotopes in the aged mixture and print the results. At this point, the user will have the opportunity to specify the binning structure for bremsstrahlung radiation. As of this version, *gsource* provides a measured bremsstrahlung spectrum due to beta decay of  $^{234m}\text{Pa}$ , scaled to the concentration of  $^{234m}\text{Pa}$  present and rebinned as specified by the user. In general, binned data will have the following options:

Select binning options for Bremsstrahlung:

- 1) Default Binning
- 2) Read from a file
- 3) Equal spaced bins
- 4) Proportional to Energy Width
- 5) TART 65 bins (NaI)
- 6) MORSE 35 bins (NaI)
- 7) GADRAS 1000 bins (HPGe)

Select (1-7):

Option 1, default binning, is a bin structure compiled into the application. This is found in source file *defaultbins.cc* and may be customized for local installations. Option 2, read from a file, reads bin boundaries from a file. A default filename is suggested, depending on the quantity being binned. Option 3 creates a specified number of bins with specified maximum and minimum energy boundaries. Option 4 does the same, but varies the bin width in proportion to the function  $k_1 + k_2 E^{1/2}$ , with  $k_1$  and  $k_2$  specified by the user and  $E$  is the lower energy boundary of the bin. The remaining options are predefined bin structures. After selecting a binning option, the user will be prompted for any additional information required by that option.

At this point, the user will be given the option to bin some x-ray and gamma ray lines and keep the others. These options are presented in a menu similar to the binning options:

Select binning options for gamma lines:

- 1) Keep all lines
- 2) Bin all lines
- 3) Keep the default lines
- 4) Read the line list from a file

Select (1-4):

These options are mostly self-explanatory. Option 3, default lines, refers to a line list compiled in the application and can again be found in *defaultbins.cc*. Option 4, read from a file, reads line energies from a file. Note that the lists of line energies must exactly match the line energies listed in the database, and changes in the database must be reflected in the line energy lists.

If you elect to bin some or all of the lines, the program will restrict its output of unbinned lines to those that fall within the same energy range as the binned lines. The unscattered photon tally will also be restricted to this energy range. If you do not bin any lines, you will be prompted to specify a maximum and minimum energy.

The problem specification is completed by selecting a binning structure for binned gamma photons (but only if you selected to bin photons) and for the scattered photon tally. These selections are made in the same way the bremsstrahlung binning was selected.

At this point, *gsource* will execute and produce two output files: *output.lin* and *output.mci*. *Output.lin* contains lists of all photons lines emitted by radioisotopes in the aged sample and their originating decays. *Output.mci* contains the photon distributions and bin structures for inclusion in an MCNP input file.

## 5 Installation

The RadSrc Suite is distributed as a single archive containing the source code and isotope data necessary for operation. Upon extraction, you will have the complete distribution in a self-contained directory structure:

- `/bin`, `/lib` — Destination directory for executables and the library.
- `/src` — Source code and makefile.
- `/doc` — Documentation and destination for doxygen files.
- `/data` — Isotope database files.

To install, simply change to the source directory and type `make`. This will compile both the library and application with the GNU compiler and install them in the appropriate directory. To compile either separately, type `make gsource` or `make libradsrc`.

The provided makefile also includes flags for the Intel and Portland Group compilers. To compile using either of these compilers, include the target `intel` or `portland` on the command line.

If you have doxygen installed on your system, type `make docs` to generate the html documentation. A file that redirects to the main page is provided in the `doc` directory for your convenience.

To accommodate applications that cannot specify the location of the isotope data base, the RadSrc Suite will check environment variables for the correct path. Since the RadSrc library can load multiple databases, each database parser will look for an environment variable specific to the parser. All parsers will fall back to the `RADSRC_DATA` environment if the parser-specific variable is not set. The GAMGEN legacy database parser will first check `RADSRC_LEGACYDATA`, then `RADSRC_DATA` to locate the database. If neither variable is set or no database is found at that location, the library will attempt to load from a subdirectory in the current directory called “data”.

You may configure *libradsrc* to use extended-precision arithmetic when calculating isotope concentrations. To do accomplish this compile with `USE_HIGH_PRECISION` defined, or edit `porting.h`. You may also use an external library by editing `porting.h`. Then, `HighPrecisionType` must be a typedef to a class that supports basic arithmetic operators and the exponential function. Note that the *gsource* application currently only links with the double precision library.

## References

- [1] “MCNPX Version 2.5.0 User’s Manual,” LA-CP-05-0369, Los Alamos National Laboratory (2005).
- [2] S. Agostinelli, *et. al.*, “GEANT4 — a simulation toolkit,” Nucl. Inst. Meth. A **506**, 250-303 (2003).
- [3] R. Buck, *et. al.* “A Multiparticle Monte Carlo Transport Code, User’s Manual, Fifth Edition,” UCRL-TM-202590, Lawrence Livermore National Laboratory (2002).

- [4] T. Gosnell, “Automated calculation of photon source emission from arbitrary mixtures of naturally radioactive heavy nuclides,” Nucl. Inst. Meth. A **299**, 682-686 (1990).
- [5] K. S. Krane, **Introductory Nuclear Physics**. John Wiley & Sons:New York (1988).
- [6] H. Bateman, “Solution of a system of differential equations occurring in the theory of radioactive transformation,” Proc. Cambridge Phil. Soc. **15**, 423-427 (1910).
- [7] D. S. Pressyanov, “Short solution of the radioactive decay chain equations,” Am. J. Phys. **70**(4), 444-445 (2002).

## A Library API

This appendix describes the *libradsrc* API. *Libradsrc* provides a simplified API for applications solely interested in computing and sampling photon distributions. A full-featured C++ API is also provided for complete access to the isotope database and decay product and photon calculations.

Note that all C++ functions are contained within the **radsrc** namespace.

### A.1 Error Handling

The library declares the **CRadSourceException** class for error handling. There are no subtypes of this class at this time. This class is thrown in the following situations:

- An unrecognized database type is requested, or a database parser is requested to parse a foreign database type. (Currently, only the GAMGEN legacy database is supported).
- The input mixture units are not `ATOMIC_FRACTION`.
- No database is loaded at the time calculations are performed.
- If you call an accessor method in an uninitialized **CGammaEntry** object or dereference an invalid **CPhotonIterator**.

### A.2 Caveats

Please be aware of the following issues.

ASCII to floating-point conversion routines differ from machine to machine. As a result, photon energies are known to be slightly different on different architectures, despite being constant values that are never computed.

Finite precision can cause the concentration of some daughter products to be negative at very short times relative to their half-lives.

### A.3 FORTRAN/C/C++ Monte Carlo Interface

In these examples,

```

LOGICAL SUCCESS
INTEGER*8 HANDLE
CHARACTER*n FILENAME, CONFIGSTRING
INTEGER LENGTH, Z, A, M, N, NMAX
DOUBLE PRECISION CONCENTRATION, E, AGE, LINES[2][NMAX], FOURV[4], DRNG
REAL*4 FRNG
INTRINSIC/EXTRINSIC FRNG, DRNG

```

Initialization will generally follow one of the following forms:

FORTRAN:

- Programmatically set the initial composition and final age.

```

CALL RSNEWSOURCE(HANDLE)
CALL ADDISOTOPE(HANDLE,92,238,0,100D0)
SUCCESS = MIX(HANDLE,25D0)

```

- Load the configuration from a file.

```

CALL RSNEWSOURCE(HANDLE)
SUCCESS = RSLOADCONFIG(HANDLE,'config.txt')

```

- Concatenate a series of character strings into a single string containing the configuration.

```

CALL RSNEWSOURCE(HANDLE)
DO
  CALL RSADDCONFIG(HANDLE,CONFIGSTRING)
END DO
SUCCESS = RSSOURCECONFIG(HANDLE)

```

C++:

- Programmatically set the initial composition and final age.

```

CRadSource* handle = newSource();
addIsotope(HANDLE,92,238,0,100.0)
bool success = mix(HANDLE,25.0)

```

- Load the configuration from a file.

```

Handle handle = newSource();
bool success = loadConfig(handle,"config.txt")

```

- Concatenate a series of character strings into a single string containing the

```

Handle handle = newSource();
while(configstring) {
  addConfig(handle,configstring);
}
bool success = sourceConfig(handle)

```

No explicit C API is provided. Instead, C programs should call the FORTRAN API functions (with trailing underscores) using the prototypes provided.

### A.3.1 Create a new radiation decay problem.

Returns or sets an 8-byte buffer as the problem handle. Multiple independent problems can be created, and are accessed via this handle.

```
static CRadSource * CApi::newSource (void)
void rsnewsources_ (char *pHandle)

CALL RSNEWSOURCE (HANDLE)
```

### A.3.2 Create and execute a new decay calculation problem.

This routine loads configuration information from a file. The filename may be either a CHARACTER variable or string literal. If the filename is the empty string, the library checks the RADSRC\_CONFIG environment variable for the filename. If the file is successfully parsed, the problem is set up and the aged mixture is calculated. The function returns true is successful, false if failed.

```
static int CApi::loadConfig (CRadSource *pRadSource, const std::string &filename)
int rsloadconfig_ (char *pHandle, char *ptr, int len)

SUCCESS = RSLOADCONFIG (HANDLE, FILENAME)
SUCCESS = RSLOADCONFIG (HANDLE, 'filename.txt')
SUCCESS = RSLOADCONFIG (HANDLE, '') for default location
```

### A.3.3 Add an isotope to the input mixture.

Input parameters are atomic number, atomic mass, metastable state, and atomic fraction in percent.

```
static void CApi::addIsotope (CRadSource *pRadSource, int z, int a, int m, double perc)
void rsaddisotope_ (char *pHandle, const int &z, const int &a, const int &m, const double &perc)

CALL RSADDISOTOPE (HANDLE, Z, A, M, CONCENTRATION)
```

### A.3.4 Get the number of discrete photons.

```
static int CApi::nLines (const CRadSource *pRadSource)
int rsnlines_ (const char *pHandle)

N = RSNLINES (HANDLE)
```

### A.3.5 Get the first nmax (energy,intensity) discrete photon entries.

Parameters are a 2-by-NMAX double precision array, and NMAX, the maximum number of entries to return. Entries are returned in sorted order.

```
static void CApi::getLines (CRadSource *pRadSource, double lines[][2], int nmax)
void rsgetlines_ (char *pHandle, double lines[][2], const int &nmax)

CALL RSGETLINES (HANDLE, LINES, NMAX)
```

### A.3.6 Get a random photon energy in keV.

Sample a energy from the photon distribution using the random number generator provided. Note that the FORTRAN and C interfaces have different function names for single and double precision random number functions.

```
static double CApi::getPhoton (const CRadSource *pRadSource, double(*prng)(void))
static double CApi::getPhoton (const CRadSource *pRadSource, float(*prng)(void))

double rsgetphoton_ (const char *pHandle, double(*prng)(void))
double rsgetrphoton_ (const char *pHandle, float(*prng)(void))

E = RSGETPHOTON (HANDLE, DRNG)
E = RSGTRPHOTON (HANDLE, FRNG)
```

### A.3.7 Get a random 4-vector E,px,py,pz in natural units (keV).

Sample an isotropic four-vector from the photon distribution using the random number generator provided. Note that the FORTRAN and C interfaces have different function names for single and double precision random number functions.

```
static void CApi::get4V (const CRadSource *pRadSource, double e[4], double(*prng)(void))
static void CApi::get4V (const CRadSource *pRadSource, double e[4], float(*prng)(void))

void rsget4v_ (const char *pHandle, double e[4], double(*prng)(void))
void rsgetr4v_ (const char *pHandle, double e[4], float(*prng)(void))

CALL RSGET4V (HANDLE, FOURV, DRNG)
CALL RSGTR4V (HANDLE, FOURV, FRNG)
```

### A.3.8 Store a summary into a character variable.

Writes the input and output mixtures to a STL string, char array, or CHARACTER variable. Note that in the C interface, the third parameter is the buffer length. The function returns the number of characters placed in the buffer. The string is *not* nul-terminated.

```
static std::string CApi::getReport (const CRadSource *pRadSource)
int rsgetreport_ (const char *pHandle, char *ptr, int len)

LENGTH = RSGETREPORT (HANDLE, BUFFER)
```

### A.3.9 Add to the growing string of configuration information.

Input may be either a CHARACTER variable or a string literal.

```
static void CApi::addConfig (CRadSource *pRadSource, const std::string &input)
void rsaddconfig_ (char *pHandle, char *ptr, int len)

CALL RSADDCONFIG (HANDLE, CONFIGSTRING)
CALL RSADDCONFIG (HANDLE, 'U238 100')
```

### A.3.10 Parse the configuration information and perform the calculations.

The function parses the configuration information provided by `addConfig()`. If successful, it sets up the problem and ages the mixture. It returns true if successful, false if failed.

```
static int CApi::sourceConfig (CRadSource *pRadSource)
bool rssourceconfig_ (char *pHandle)

SUCCESS = RSSOURCECONFIG(HANDLE)
```

### A.3.11 Age the input mixture.

This function ages the mixture set by `addIsotope`. The parameter is the age in years. Returns true if successful, false if failed.

```
static int CApi::mix (CRadSource *pRadSource, double age)
int rsmix_ (char *pHandle, const double &age)

SUCCESS = RSMIX(HANDLE,AGE)
```

### A.3.12 Sort the photon list.

Sorts the photon list by ascending energy or descending intensity. Parameter is 1 for energy and 2 for intensity.

```
static void CApi::sort (CRadSource *pRadSource, int field)
void rssort_ (char *pHandle, const int &field)

CALL RSSORT(HANDLE,1) for energy
CALL RSSORT(HANDLE,2) for intensity
```

## A.4 Class CIsotope

The **CIsotope** class is the fundamental identifier for isotopes in the library. Nuclear isomers are distinguished by a metastable state number. The **CIsotope** class also possesses convenient conversion functions to and from isotope names.

### A.4.1 Construct a CIsotope

```
CIsotope ()
CIsotope (int z, int a, int mm=0)
```

### A.4.2 get Z, A, and metastable level

```
int getAtomicNumber (void) const
int getMassNumber (void) const
int getMetastableNumber (void) const
```

### A.4.3 Obtain the canonical name of the isotope.

These methods create the canonical name of the isotope. Names are of the form Zzz-AAAmN, with a maximum size of 9 characters. Invalid isotopes are named "H-0".

```
void toString (char *str) const  
void toString (std::string &str) const  
  
std::string toString () const
```

### A.4.4 Parse variations of the isotope name.

These methods define the **CIsotope** by parsing a string. The '-' is optional but must not be whitespace. Capitalization is also ignored.

```
CIsotope & fromString (const char *str)  
CIsotope & fromString (const std::string &str)
```

### A.4.5 Is this a valid isotope? (conversion from strings can fail)

If the **CIsotope** is not initialized or **fromString** fails, this method will return true.

```
bool isValid (void) const
```

## A.5 Class CDatabaseManager

This class creates isotope databases from database-specific routines and classes. **CIsotopeDatabase** and **CDatabaseManager** provide a uniform interface for accessing isotope information regardless of the source, storage, or formatting of the underlying data. The **CDatabaseManager** class is a singleton.

### A.5.1 Enumerations

```
enum DatabaseType { LEGACY, ENSDF, ENSDF_ERRATA }
```

### A.5.2 Typedef

```
typedef std::pair< int, std::string > DatabaseIdentifier
```

### A.5.3 Obtain a pointer to the library's CDatabaseManager.

A single **CDatabaseManager** object manages all the databases in the library. This function will return a pointer to it.

```
static CDatabaseManager * getDatabaseManager (void)
```

#### A.5.4 Obtain a pointer to a particular isotope database.

This function loads an isotope database and returns a pointer to it if successful. This first parameter is an enum of type **DatabaseType**. Currently only **LEGACY**, the GAMGEN database format, is supported. This parameter selects which database parser is to be used. The second parameter is an identifier to a specific database. The meaning of this parameter is defined by the selected parser, but is typically one or more filenames. Databases with the same parser and identifier are unique and need only be loaded once.

### A.6 Class CIsotopeDatabase

The **CIsotopeDatabase** class maintains a mapping from **CIsotope** to **CIsotopeData**, and provides methods and iterators to access isotope data in the mapping.

#### A.6.1 Typedefs

```
typedef std::map< CIsotope, CIsotopeData * > IsotopeList
typedef std::map< CIsotope, CIsotopeData * >::const_iterator IsotopeListIterator
```

#### A.6.2 Get the library's isotope database manager object.

```
static CIsotopeDatabase * getIsotopeDatabase (int type, std::string info="")
```

#### A.6.3 Obtain information on an isotope.

Returns a pointer to a **CIsotopeData** object if the database contains an entry for the isotope, or 0 if not.

```
const CIsotopeData * getIsotopeData (const CIsotope &iso) const
```

#### A.6.4 Check if an isotope is present in the database.

```
bool hasIsotopeData (const CIsotope &iso) const
```

#### A.6.5 Iterators for accessing isotope data.

These methods return iterators to access isotope data for each isotope in the database.

```
IsotopeListIterator isotopesBegin (void) const
IsotopeListIterator isotopesEnd (void) const
```

#### A.6.6 Get the number of isotopes in database.

```
int getNIsotopes (void) const
```

## A.7 Class CIsotopeData

The **CIsotopeData** class encapsulates the basic isotope constants and a listing of possible decays. Decay entries are distinct even if they ultimately decay to the same daughter isotope. For example, multiple beta decay branches to different nuclear states, which then immediately decay, can each have an entry in the database with unique associated photon emissions.

### A.7.1 Typedef

```
typedef std::vector< CDecayMode >::const_iterator DecayIterator
```

### A.7.2 Get the decay rate of a particular branch.

Units are disintegrations per second. The parameter is either a branch number starting with zero or **DecayIterator**. Units are in disintegrations per second.

```
double getDecayRate (int branch) const  
double getDecayRate (const DecayIterator &it) const
```

### A.7.3 Get the decay rate of the isotope

Units are disintegrations per second.

```
double getDecayRate (void) const
```

### A.7.4 Decay branch iterators

```
DecayIterator decaysBegin (void) const  
DecayIterator decaysEnd (void) const
```

### A.7.5 Get the number of decay branches.

```
int getNDecayModes (void) const
```

### A.7.6 Get the isotope which this entry describes.

```
const CIsotope & getIsotope (void) const
```

### A.7.7 Get the canonical name of this isotope.

```
const char * getName (void) const
```

### A.7.8 Get the standard average atomic mass for the isotope.

Units are in grams.

```
double getAtomicMass (void) const
```

### A.7.9 Get the halflife of the isotope.

Units are in seconds.

```
double getHalflife (void) const
```

## A.8 Class CDecayMode

The **CDecayMode** class encapsulates the information about a particular decay. This includes the branch fraction, the discrete photon lines, and the average bremsstrahlung spectrum.

### A.8.1 Enumerations

```
enum DecayType { UNSPECIFIED, ALPHA, BETA_GENERIC, BETA_MINUS, BETA_PLUS,  
ELECTRON_CAPTURE, INTERNAL_TRANSITION, ALPHANEUTRON, BETANEUTRON  
}
```

### A.8.2 Typedefs

```
typedef std::vector< CPhoton >::const_iterator PhotonIterator  
typedef std::vector< double >::const_iterator BremBoundaryIterator  
typedef std::vector< double >::const_iterator BremIntensityIterator
```

### A.8.3 Get the decay type.

This function returns the type of decay. The usefulness of this value is entirely dependent upon the quality of the underlying source of the decay information and the routine that parses it.

```
int getDecayType (void) const
```

### A.8.4 Get the daughter isotope of this particular decay.

The method returns the daughter isotope of the decay, which may be a specific isomer.

```
const CIsotope & getDaughter (void) const
```

### A.8.5 Get the branch fraction of this particular decay.

This method returns the branching ratio of this particular decay.

```
double getBranchFraction (void) const
```

### A.8.6 Get number of photons produced in decay.

```
int getNPhotons (void) const
```

#### A.8.7 Obtain iterators for the discrete photons produced by this decay.

**PhotonIterator beginPhotons** (void) const

**PhotonIterator endPhotons** (void) const

#### A.8.8 get number of brem bins

int **getNBremBins** (void) const

#### A.8.9 Obtain iterators for the bremsstrahlung energy bin boundaries.

**BremBoundaryIterator beginBremBoundaries** (void) const

**BremBoundaryIterator endBremBoundaries** (void) const

#### A.8.10 Obtain iterators for the bremsstrahlung bin intensities.

**BremIntensityIterator beginBremIntensities** (void) const

**BremIntensityIterator endBremIntensities** (void) const

### A.9 Class CPhoton

The **CPhoton** class is a database entry for a photon. It contains the photon energy, the probability of emission, and relative uncertainty in that probability.

#### A.9.1 Get the photon energy.

Units are keV.

double **getEnergy** (void) const

#### A.9.2 Get the emission probability.

Probability is per decay.

double **getFraction** (void) const

#### A.9.3 Get the relative error in the emission probability.

Error is  $\frac{\Delta F}{F}$ .

double **getError** (void) const

### A.10 Class CDecayComputer

The **CDecayComputer** class stores the input and aged mixtures, and retains the time dependence of the aged concentrations in a mapping of **CIsotope** to **CBatemanSolution**.

#### A.10.1 Convert input to canonical units.

At this time, the only valid unit selection is **ATOM\_FRACTION**. This also the default selection for the object and this call is optional.

```
void normalizeInputUnits (void)
```

#### A.10.2 Append a radioisotope to the input mixture list.

The parameters are a fully constructed **CIsotope** class and the quantity of that isotope in unspecified units. The units will later be defined with a call to **normalizeInputUnits**.

```
void addInputItem (const CIsotope &iso, double amount)
```

#### A.10.3 Reset the object.

This method clears all data and settings in the object, except the database selection.

```
void clear ()
```

#### A.10.4 Compute the isotope concentrations at a particular age.

This method causes the decay chain to be traversed and the time dependence of each isotope in the chain is computed. These are evaluated at the specified age to produce the aged mixture. The parameter is the age in years.

```
CIsotopeMixture & ageMixture (double age)
```

#### A.10.5 Get the detailed solution.

Returns a mapping of **CIsotope** to **CBatemanSolution**, which contains the full time dependence and parent-age of every isotope in the decay chain.

```
const std::map< CIsotope, CBatemanSolution > & getFullSolution (void) const
```

#### A.10.6 Get the detailed solution for an Isotope.

Returns a **CBatemanSolution** object, which contains the full time dependence and of the isotope.

```
const CBatemanSolution * getSolution (const CIsotope &isotope) const
```

#### A.10.7 Get mixture at a particular time.

Returns a mapping of **CIsotope** to double, giving the concentration of each isotope in the aged mixture. The units are in atomic fraction.

```
const CIsotopeMixture & getAgedMixture (void) const
```

#### A.10.8 Get the initial mixture.

Returns a mapping of **CIsotope** to double, giving the concentration of each isotope in the initial mixture. The units are in atomic fraction.

```
const CIsotopeMixture & getInputMixture (void) const
```

#### A.10.9 Look up some data in the current database.

Shortcut to obtain an isotope data entry from the database currently being used by this **CDecayComputer** object.

```
const CIsotopeData * getIsotopeData (const CIsotope &isotope) const
```

#### A.10.10 Get the current isotope database.

Get the current isotope database being used by this **CDecayComputer** object.

```
const CIsotopeDatabase * getIsotopeDatabase (void) const
```

#### A.10.11 Set the database to be used.

Set the isotope database to be used for future calculations. This action resets the object as indicated in the method **clear**.

```
void initialize (const CIsotopeDatabase *pisotopedb)
```

### A.11 Class **CBatemanSolution**

The **CBatemanSolution** stores the coefficients and decay constants for each term in the solution for a particular isotope that may appear in the aged mixture. It also retains a listing (as an STL set) of ultimate parent radioisotopes in the initial mixture contributing the isotope.

#### A.11.1 Get the isotope for which this object is a solution.

```
const CIsotope & forIsotope (void) const
```

#### A.11.2 Get a list of initial isotope parents.

Get a list (as an STL set) of radioisotopes in the initial mixture that eventually decayed into this isotope.

```
const std::set< CIsotope > & getChainParents (void) const
```

### A.12 Class **CIsotopeMixture**

This class is a mapping from **CIsotope** to double, providing the concentrations of all the isotopes in the list.

### A.12.1 Compute the average atomic mass.

Computes the average atomic mass of the mixture, thus giving grams/mol.

```
double computeAverageMass (void) const
```

### A.12.2 Set the isotope database

Sets the isotope database to be used for information about isotopes in this mixture.

```
void setDatabase (const CIsotopeDatabase *)
```

## A.13 Class **CPhotonComputer**

The **CPhotonComputer** class stores lists of photon energy and intensity and maintains the association between a photon and its emitting isotope(s) in an aged mixture. A **CPhotonComputer** is permanently associated with a **CDecayComputer** object, and its associated isotope database. **CPhotonComputer** provides an iterator class to access the sorted photon list.

### A.13.1 Enumerations

```
enum { ENERGY = 0, INTENSITY = 1 }  
enum { PERMOLE, PERGRAM }  
enum BinSubject { BIN_BREM, BIN_GAMMA }
```

### A.13.2 Typedefs

```
typedef std::map< CIsotope, double > IsotopeMixture
```

### A.13.3 Create and sort the list of emitted discrete photons.

Computes the discrete lines emitted by the decay of elements present in the aged mixture in the associated **CDecayComputer** object. This function may be called again to change the sort order without repeating the calculation. The parameter is one of the enums **ENERGY** (ascending energy) or **INTENSITY** (descending intensity).

```
void computeGammas (int sortparam=ENERGY)
```

### A.13.4 Get iterators for the lists of photons

These methods return begin and end iterators for the complete list of photons, and the subset list of photons, respectively.

```
CPhotonIterator beginGammas (void) const  
CPhotonIterator endGammas (void) const  
CPhotonIterator beginSelectedGammas (void) const  
CPhotonIterator endSelectedGammas (void) const
```

#### A.13.5 Get the number of discrete lines in the list of photons.

These methods return the size of the complete list of discrete photons, and the size of the subset list, respectively.

```
int getNGammas (void) const  
int getNSelected (void) const
```

#### A.13.6 Set the bin boundaries.

These methods set the bin boundaries of the bremsstrahlung and non-selected (binned) photon lines. The first parameter is one of the enums **BIN\_BREM** or **BIN\_GAMMA**. The second parameter may be either an STL vector of doubles listing the bin boundary energies, or an array of doubles. In the later case, the length of the array must be passed as the third argument.

```
void setBinning (BinSubject what, const std::vector< double > &v)  
void setBinning (BinSubject what, const double *, int)
```

#### A.13.7 Select a subset of the discrete lines and bin the rest.

These methods subset the list of photons according to a list of desired energies. Photons that are not in the list of desired energies are combined into a distribution of binned intensities. Energies may be provided as either an STL vector of doubles, or an array of doubles.

Care must be taken to ensure that the energies in the list and the energies in the isotope database are identical in the machine's native representation.

```
void selectGammas (const std::vector< double > &v)  
void selectGammas (const double *lines=0, int n=0)
```

#### A.13.8 Get the bremsstrahlung binned data.

Returns a **CBinnedData** object which contains the bin boundaries and bin intensities of the bremsstrahlung distribution.

```
const CBinnedData & getBrem (void) const
```

#### A.13.9 Get the binned lines data.

Returns a **CBinnedData** object which contains the bin boundaries and bin intensities of the **non**-selected discrete lines.

```
const CBinnedData & getBinnedGammas (void) const
```

#### A.13.10 Sample the photon distributions.

These methods sample the combined discrete and bremsstrahlung intensity distribution. The first two methods return a photon energy in keV, while the second two methods fill an array of energy and momentum values in natural units (keV). In the latter case, the first parameter is an array of four doubles  $\{E, p_x, p_y, p_z\}$ . All four methods require a pointer to a function returning either a single- or double- precision random number in the range  $[0,1)$ .

```
double getPhoton (double(*rng)(void)) const  
double getPhoton (float(*rng)(void)) const  
void getFourVector (double e[4], double(*rng)(void)) const  
void getFourVector (double e[4], float(*rng)(void)) const
```

#### A.14 Class CPhotonIterator

The **CPhotonIterator** class combines information from **CPhotonComputer** and **CDecayComputer** to provide complete information about emitted photon lines. **CPhotonIterator** follows const forward iterator semantics. **CPhotonIterator** dereferences to a const **CGammaEntry** object.

N.B. References the **CGammaEntry** are valid only while while the **CPhotonIterator** points to it. If the application requires the **CGammaEntry** to persist then a copy should be made.

#### A.15 Class CGammaEntry

The **CGammaEntry** class encapsulates all the information known about a discrete photon line, including its origins in the decay chain.

##### A.15.1 Typedefs

```
typedef std::set< std::pair< CIsotope, CIsotope > > DecayList  
typedef std::set< CIsotope > ParentList
```

##### A.15.2 Get the isotope in the initial mixture that produces this line.

This method returns a list (as an STL set) of all the isotopes in the initial mixture that decayed into an isotope that subsequently emitted this line.

```
const ParentList & getChainParentIsotopes (void) const
```

##### A.15.3 List the decays that produce a line.

This method returns a list (as an STL set of isotope pairs) of parent and daughter isotopes that produce this line.

```
const DecayList & getDecays (void) const
```

#### A.15.4 List the isotopes that decayed and emitted a line.

This method returns a list (as an STL set) of all the isotopes that emit this line in the process of, or as a result of, decaying.

```
const ParentList & getParentIsotopes (void) const
```

#### A.15.5 Get a formatted list decays and ultimate parent isotopes of a line.

This method returns an STL string containing a list of isotopes in the initial mixture that eventually produce this discrete line. These isotopes are printed in square brackets []. It then lists the specific decays, parent to daughter, separated by arrows, ->.

```
std::string getParentDescription (void) const
```

#### A.15.6 Get the photon energy.

This method returns the photon energy in keV.

```
double getEnergy (void) const
```

#### A.15.7 Get the photon intensity

This method returns the photon intensity in the current units. (default: photons/sec/gram of input mixture) **HighPrecisionType** is defined in porting.h at compile time by the user.

```
HighPrecisionType getIntensity (void) const
```

### A.16 Class CBinnedData

**CBinnedData** is essentially a structure describing binned data. Its members are STL vectors of doubles or **HighPrecisionType** containing the bin energy boundaries, the bin intensities, the cumulative intensity and total intensity.

#### A.16.1 Members

```
std::vector< double > m_energy  
std::vector< HighPrecisionType > m_intensity  
std::vector< HighPrecisionType > m_cumulative  
  
HighPrecisionType m_sum
```